Learning From Failure: Integrating Negative Examples when Fine-tuning Large Language Models as Agents

Renxi Wang[1,2] **Haonan Li**[1,2] **Xudong Han**[1,2]
Yixuan Zhang[1,2] **Timothy Baldwin**[1,2,3]
[1]LibrAI [2]MBZUAI [3]The University of Melbourne
{renxi.wang,haonan.li,xudong.han,yixuan.zhang,timothy.baldwin}@mbzuai.ac.ae

Abstract

Large language models (LLMs) have achieved success in acting as agents, which interact with environments through tools like search engines. However, LLMs are not optimized specifically for tool use during training or alignment, limiting their effectiveness as agents. To resolve this problem, previous work has collected interaction trajectories between GPT-4 and environments, and fine-tuned smaller models with them. As part of this, the standard approach has been to simply discard trajectories that do not finish the task successfully, which, on the one hand, leads to a significant waste of data and resources, and on the other hand, has the potential to limit the possible optimization paths during fine-tuning. In this paper, we contend that large language models can learn from failures through appropriate data cleaning and fine-tuning strategies. We conduct experiments on mathematical reasoning, multi-hop question answering, and strategic question answering tasks. Experimental results demonstrate that compared to solely using positive examples, incorporating negative examples enhances model performance by a large margin.[1]

1 Introduction

An agent is a model which has the ability to interact with an environment, make decisions, and achieve predefined goals (Wooldridge, 1999). Early work used rule-based or template-based systems to complete tasks in narrow and specialized domains (Green Jr et al., 1961; Weizenbaum, 1966). Recent work has taken powerful LLMs such as GPT-4 (Achiam et al., 2023) ans used them as the core of an agent system to process information and make decisions (Gravitas, 2024; Yoheinakajima, 2024). This line of work has resulted in agent systems that are able to do much more complex and general tasks.

However, these models rely on closed models through paid APIs, raising concerns about cost, latency, and reproducibility. Additionally, existing LLMs were not developed for agent use cases (e.g., generating actions), and few-shot prompting offers only limited learning support (Chen et al., 2023).

Subsequent work has explored fine-tuning LLMs as agents, typically in three stages: data collection, fine-tuning, and inference (Chen et al., 2023; Zeng et al., 2023; Yin et al., 2023; Qiao et al., 2024). At the data collection stage, a powerful LLM such as GPT-4 is employed to interact with the environment, and the LLM-generated outputs and environment observations are collected as trajectories. In the fine-tuning stage, smaller models are fine-tuned using only successful trajectories. The fine-tuned models then serve as the agent's core during inference, demonstrating enhanced tool-using and decision-making capabilities, sometimes even surpassing the performance of the original LLM.

To ensure the agent is being optimized appropriately, previous work has simply discard trajectories that do not successfully complete the task, using only successful trajectories in the fine-tuning stage. However, in tasks demanding intricate planning, reasoning, or tool usage, the volume of discarded negative samples can exceed 60%, leading to substantial data and computational resource wastage.

In this paper, we explore two key questions: (1) Can LLMs learn from negative examples through fine-tuning? and (2) How can we optimize the use of negative examples to enhance agent performance? To address the first question, we fine-tune LLMs with a mix of positive and negative examples, and observe that incorporating negative examples generally yields benefits. For the second question, we introduce a negative-aware training (NAT) approach that explicitly instructs the model to differentiate between correct and incorrect interactions. Our experiments demonstrate that NAT outperforms traditional methods that solely use pos-

[1]Code and data are available at: https://github.com/Reason-Wang/NAT.

itive examples or naively combine positive and negative examples. In addition, we conduct extensive experiments to analyze the learned agents' behavior when fine-tuning with negative examples. We find that LLMs implicitly learn a lot from negative samples when combined with positive ones.

Our contributions can be summarized as follows:

- We demonstrate the value of negative trajectories and introduce a negative-aware training approach, allowing LLM-based trained agents to effectively learn from both positive and negative examples.
- We validate the broad applicability and effectiveness of learning from negative examples, and show that it enables models to acquire information akin to positive examples across various tasks and prompting strategies.

2 Related Work

2.1 Fine-tuning LLMs as Agents

Previous work on language agents has taken a powerful LLM as the core of the agent system without fine-tuning (Sumers et al., 2023; Wu et al., 2023; Ruan et al., 2023; Zhao et al., 2023). However, LLMs are optimized to generate natural language. To make them capable of using tools and making decisions, current work typically collects trajectories generated by GPT-3.5/4, then uses only successful trajectories to fine-tune a smaller LLM. Zeng et al. (2023) collect trajectories generated by GPT-4 on AgentBench (Liu et al., 2023) tasks, and only keep samples that received the best rewards. Chen et al. (2023) collect trajectories on question answering tasks and fine-tune models with samples that correctly answered the question. Liu et al. (2024) propose a memory-enhanced agent framework and a complex filtering mechanism to collect fine-tuning datasets. Qiao et al. (2024) divide an agent into sub-agents with different functions. They then synthesize trajectories for these agents respectively. However, they still only use samples with the best rewards. Although a simple ablation study is done by Zeng et al. (2023), none of them has investigated the effectiveness of negative samples in detail. Although not directly comparable, in Table 1, we compare the methods of these papers with the ours on several main benchmarks.

2.2 Learning from Negative Trajectories

Learning from negative trajectories can be divided into prompting-only and fine-tuning-based methods. Prompting-based methods enable LLMs to summarize experiences from previous mistakes without updating parameters. Madaan et al. (2023) use LLMs to first generate an output and then refine the output iteratively, while Shinn et al. (2023) employ an evaluator to provide external feedback. Zhao et al. (2023) let the agent compare successful and unsuccessful trajectories, and extract insights based on comparison. The success of these methods relies on the quality of the evaluator used to analyze the trajectories. The performance of fine-tuning-based methods is less predictable since model weights are updated, and less work has been done on this. Li et al. (2023) propose a two-stage training paradigm to capture knowledge from negative samples. However, their method focuses on Chain-of-Thought prompts and is complex since multiple models are fine-tuned. Our work focuses on the fine-tuning-based method and is both much simpler and more effective.

Model	GSM8K	SVAMP	HotpotQA
AutoAct-7B	–	–	29.2
AgentLM-7B	24.6	–	22.3
Lumos-O-7B	**50.5**	**65.5**	24.9
Lumos-I-7B	47.1	63.6	<u>29.4</u>
NAT-7B	<u>49.1</u>	<u>64.4</u>	**29.8**
CodeLlama-13B	<u>36.1</u>	<u>60.0</u>	–
AgentLM-13B	32.4	–	**29.6**
NAT-13B	**53.8**	**70.6**	**29.6**

Table 1: Comparison with methods from other papers. We report the best results reported in the corresponding papers.

3 Negative-Aware Training

3.1 Background

As shown in Figure 1, in our agent framework, the process of task resolution is delineated as follows. First, the LLM is provided with an initial prompt that outlines (a) the specific task to be addressed (for instance, *solve a mathematical problem*), (b) the tools that are permissible for task execution, and (c) the expected action space and output format (for example, *finish[N]* signifies that *N* is the final answer). See the example in Figure 2. Second, an instance is introduced, prompting the model to generate an initial series of "thoughts" and actions. Finally, during the interaction phase, the system executes the LLM-generated actions using the predefined tools, returns the resulting observations back

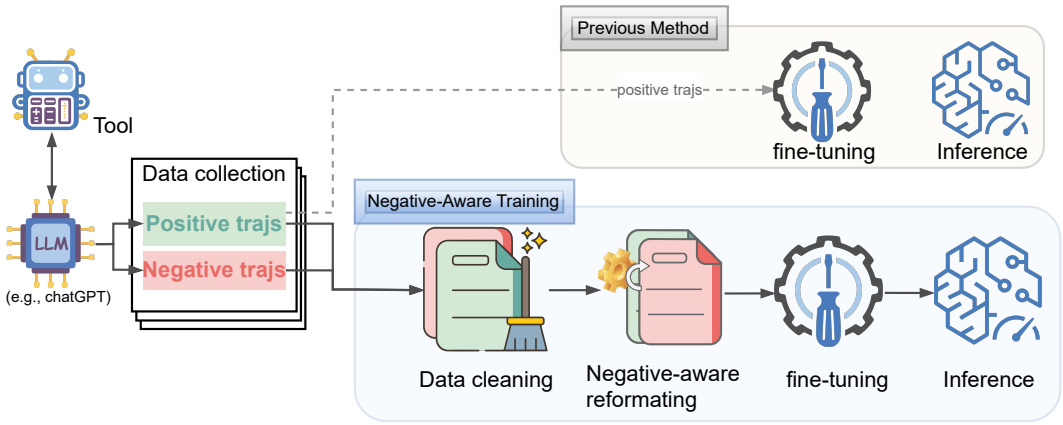

Figure 1: Illustration of negative-aware training (NAT).

to the LLM, and prompts for subsequent actions until the finished action of the task is generated, or the interaction rounds exceeds a pre-defined threshold.

Naturally, the task-solving process yields interaction trajectories between the LLM and the environment (i.e., tools in our framework). Prior research has demonstrated that fine-tuning the LLM with successful interaction trajectories can significantly enhance its problem-solving capabilities (Chen et al., 2023; Zeng et al., 2023).

In this study, we propose to use not only positive examples (i.e., trajectories that lead to successful task resolution), but also negative samples (i.e., trajectories that fail to resolve the task).

3.2 Motivation

Our idea is motivated by two considerations. First, humans learn from failure, because it provides valuable lessons that contribute to personal growth and improvement. Failure is often seen as a stepping stone to success, as it offers insights into what doesn't work and highlights areas that require change or development.

Second, even the most advanced LLMs such as GPT-4 produce unsuccessful trajectories in over 50% of attempts when acting as an agent to solve simple mathematical reasoning questions. Simply discarding these trajectories leads to a waste of computational resources, and ignores potentially valuable supervision signal.

3.3 Proposed Approach

Our method can be structured into four phases, as detailed below: (1) data collection, (2) data cleaning, (3) negative-aware reformatting, and (4) fine-tuning.

Data collection For each task, we obtain the initial questions and corresponding ground truth answers as seed data. We then use an LLM to generate trajectories three times, each with different temperatures (0.2, 0.5, and 0.7). This allows us to gather a diverse range of positive samples while experimenting with various negative sample collection strategies. In this paper, we use GPT-3.5-1106 to generate high-quality trajectories.[2]

Data cleaning We contend that certain negative samples are beneficial for fine-tuning models, while others may be harmful. Therefore, data cleaning plays a vital role in our approach. Given the variability across tasks, we adopted task-specific data-cleaning criteria and methodologies, which we detail in the experimental setup of each task.

Negative-aware reformatting Differentiating positive samples from negative samples during the agent tuning process aids in teaching the model to discern between successful and unsuccessful outcomes. We frame this as implicit contrastive learning, to help the model learn from both success and failure. Therefore, before fine-tuning, we concatenate different prompts to positive and negative trajectories, to differentiate them to the model.

Fine-tuning After negative-aware reformatting, we use the reformatted trajectories to fine-tune an LLM, which we call negative-aware training (NAT). The loss is computed only on the part of the text generated by the LLM, as seen in Figure 2 in the blue boxes. As an ablation, for all tasks, we also fine-tune the models on the dataset without negative-aware reformatting, where the posi-

[2]Although GPT-4 has the potential to produce even higher quality data, we opted for GPT-3.5 due to cost considerations.

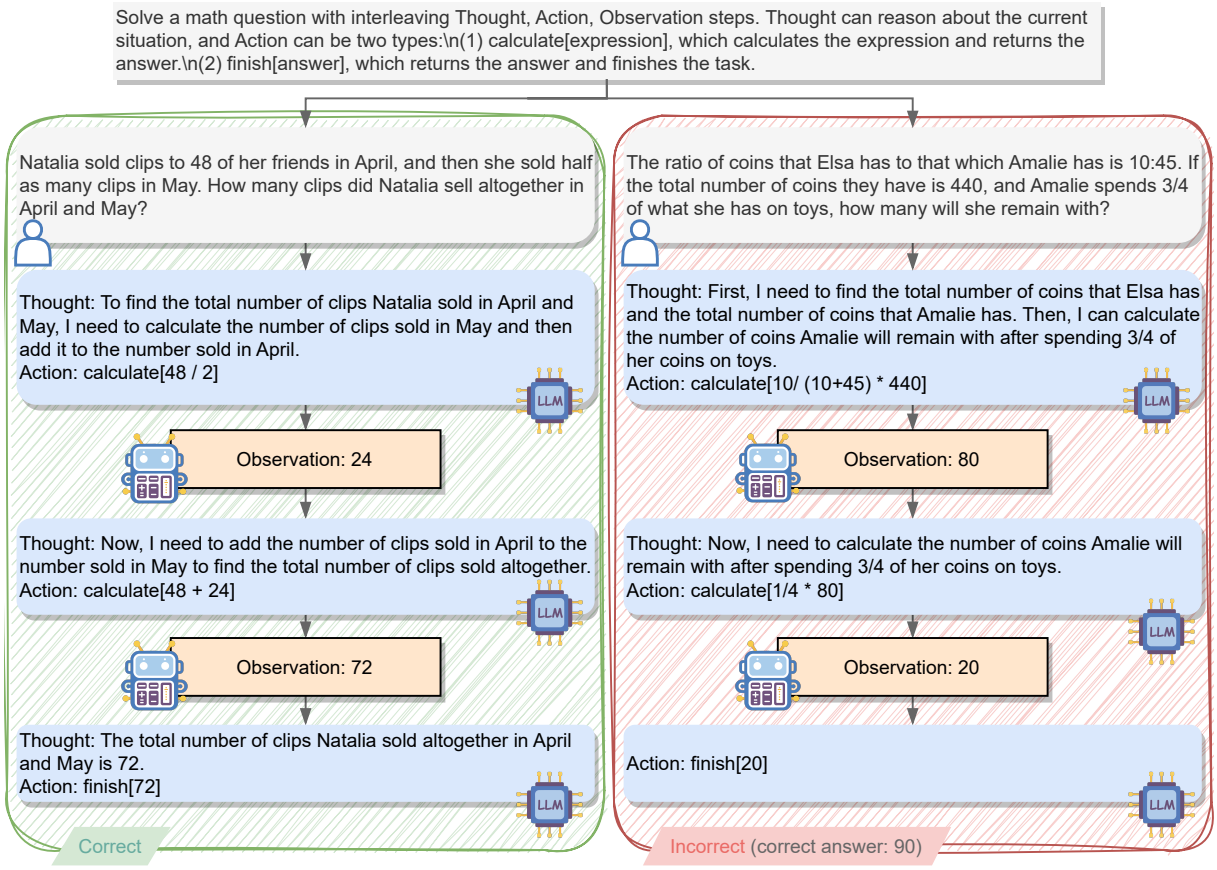

Figure 2: Example of positive (left) and negative (right) trajectories obtained from the GSM8K dataset.

tive and negative examples are concatenated with the same prompt. We name this training process negative-unaware training (NUT).

Inference During inference, we prompt the fine-tuned agent using the prompt for positive examples only.

3.4 Experimental Setup

Following prior work (Chen et al., 2023; Yin et al., 2023), we conduct experiments on mathematical reasoning, multi-hop question answering, and strategic question answering tasks, using GSM8k (Cobbe et al., 2021), HotpotQA (Yang et al., 2018), and StrategyQA (Geva et al., 2021), respectively.

Agent Framework We use `ReAct` (Yao et al., 2023), which consists of interleaving "thoughts" and actions. During inference, an observation from the environment is given to the model. The model first generates a "thought", which is plain-text reasoning based on the observation as to what it should do next. Then, it generates an action, which is what the tools use in agent scenarios. We also experiment with Chain-of-Thought (Wei et al., 2022) prompting in Section 6.

Tools For math tasks, we design a calculator implemented by SymPy (Meurer et al., 2017), which takes a math expression as input and outputs the result. For the two question-answering tasks, we design a search tool for the Serper API. It takes a search query as input and returns the Google search results. We further re-rank the search results using MPNet (Song et al., 2020) and DPR (Karpukhin et al., 2020).

Fine-tuning Setup We conduct experiments on LLaMA-2-Chat 7B and 13B models. All the models are fine-tuned for 2 epochs with a batch size of 64. We use a cosine scheduler with 3% of total steps as the warm-up. The maximum learning rate is set to 5×10^{-5}. We train the model with $4 \times$A100 GPUs with DeepSpeed ZeRO 3 stage (Rajbhandari et al., 2019).

4 Math Reasoning

4.1 Datasets

For mathematical reasoning tasks, we use a dataset of approximately 7k instances from the GSM8K training set as initial seed data, and generate three

Model	# Positive	Strategy	GSM8K	ASDiv	SVAMP	MultiArith	Average
LLama-2-7B	2k	w/o Neg	35.63	60.55	47.40	80.03	55.90
		NUT	44.43	65.69	60.40	83.05	63.39
		NAT	**46.93**	**66.93**	**60.80**	**83.89**	**64.64**
LLama-2-7B	5k	w/o Neg	45.87	68.12	58.80	83.89	64.17
		NUT	47.54	67.03	63.50	81.71	64.95
		NAT	**49.05**	**68.66**	**64.40**	**87.58**	**67.42**
LLama-2-13B	2k	w/o Neg	44.43	66.49	65.40	**84.40**	65.18
		NUT	49.43	67.72	67.60	81.37	66.53
		NAT	**50.64**	**67.92**	**68.50**	83.89	**67.74**
LLama-2-13B	5k	w/o Neg	**54.21**	**71.28**	68.30	89.26	70.76
		NUT	51.40	70.34	68.60	86.07	69.10
		NAT	53.75	70.49	**70.60**	**90.27**	**71.28**

Table 2: Overall results for math tasks. Each block is a setting with a specific model and number of positive examples. The best results are **bolded** and second best results are underlined

trajectories with GPT-3.5, as mentioned in Section 3.3. This process results in a collection of around 9k positive examples and 12k negative examples. Among the positive examples, 5k are unique, indicating that despite multiple attempts, GPT-3.5 fails to solve 2k out of the 7k original questions.

For our experiments, we incorporate 5k unique positive examples from GSM8K to emulate all available positive examples having been generated by GPT-3.5. Additionally, we created a simulated limited dataset using the 2k positive examples generated by ChatGPT. In both scenarios, we include 10k negative examples.

We evaluate different models and training strategies on four test datasets: GSM8K (Cobbe et al., 2021), a high-quality school math word problem dataset containing 1,319 examples (test set), each requiring 2–8 steps to solve; ASDiv (Miao et al., 2020), a math word problem dataset that contains 2,023 examples with diverse language patterns and problem types. SVAMP (Patel et al., 2021), a challenge set of math word problems with 1k examples based on perturbing existing datasets (Miao et al., 2020; Koncel-Kedziorski et al., 2016). MultiArith (Roy and Roth, 2015), a multi-step arithmetic problem dataset with 596 examples.

4.2 Results

Table 2 presents the overall results of the math tasks, from which we observe: (1) In the first three settings (out of four), incorporating negative examples improves model performance on almost all test sets; (2) In all four settings, models with negative-aware training (NAT) not only outperform the corresponding model trained only on positive examples,

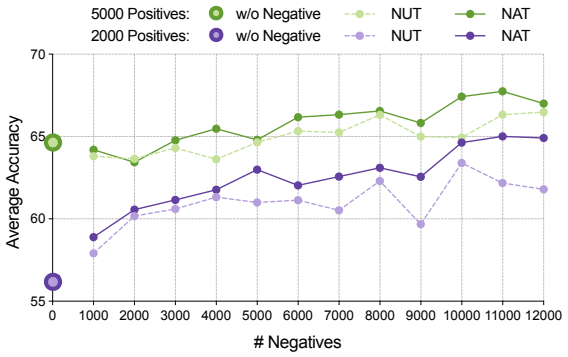

Figure 3: Performance of LLaMA-2-7B for a fixed number of positive samples and variable number of negative samples.

but also beat the same model trained with negative-unaware training (NUT); and (3) The improvement of NAT is more substantial when there are fewer positive examples or the model is smaller. Specifically, NAT achieves an 8.74 improvement when using a 7B model with 2k positive examples, and a 0.52 improvement when using a 13B model with 5k positive examples. This highlights the value of NAT in data-scarce scenarios, which is common for agent tuning.

It is worth noting that previous work (Zeng et al., 2023) has shown that including negative examples harms model performance. We believe this does not contradict our findings: as we discuss in Section 4.3, performance change is determined by the quality of the negative data.

4.3 Analysis

Table 2 showcases the capability of LLMs to learn from negative examples when fine-tuned to func-

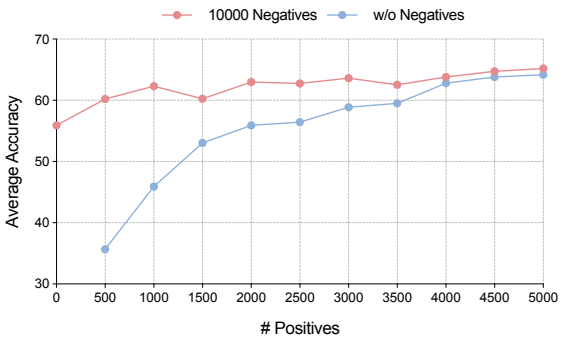

Figure 4: Performance for a fixed number of negative samples (10k) and variable number of positive samples.

Data	GSM8K	ASDiv	SVAMP	MArith	Avg
2K positive samples					
w/o Neg	35.63	60.55	47.40	80.03	55.90
NAT-low	32.98	58.72	47.60	71.64	52.74
NAT-high	**46.93**	**66.93**	**60.80**	**83.89**	**64.64**
5K positive samples					
w/o Neg	45.87	68.12	58.80	83.89	64.17
NAT-low	38.59	62.28	52.50	78.52	57.97
NAT-high	**49.05**	**68.66**	**64.40**	**87.58**	**67.42**

Table 3: LLaMA-2 7B model results trained with different quality negative data. We use 10k negative samples and experiment with 2k or 5k positive samples.

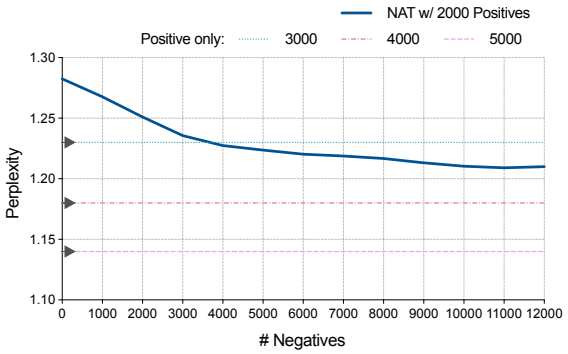

Figure 5: Perplexity for the model trained with 2k positive samples and differing numbers of negative samples. The three dashed lines are perplexity computed on models tuned with differing numbers of positive trajectories (without negatives).

tion as agents. Here, we delve into various factors that could influence the effectiveness of negative-aware training. Specifically, we seek to address the following questions: (1) Given a fixed number of positive examples, how much negative data should be used? (2) What insights does the model gain from negative trajectories? (3) Are all types of negative examples beneficial? and (4) What factors contribute to negative-aware training (NAT) outperforming negative-unaware training (NUT)?

Impact of training sample quantity Our initial analysis focuses on the influence of negative sample quantity. We maintain a constant number of positive samples at 2k or 5k, while adjusting the negative samples from 0 to 12k. The results, depicted in Figure 3, illustrate the relationship between the quantity of negative data and average math task performance. We observe a performance enhancement with an increase in negative data, which plateaus when the volume of negative samples is about 11k in both cases. Due to data availability, we did not experiment with more negatives.

Based on insights from Table 2 and Figure 3, we hypothesize that the ideal ratio of negative samples is not fixed. Instead, it is influenced by two main factors: (1) the saturation point of positive sample utility (i.e., the threshold beyond which additional positive samples cease to enhance model performance) and the actual volume of positive samples; and (2) the intrinsic quality of the negative samples.

Regarding the first point, we hypothesize that the marginal utility of negative samples diminishes as the quantity of positive samples increases. To test this point, we maintain a constant number of negative samples while varying the quantity of positive samples from 0 to 5k. As depicted in Figure 4, there is a diminishing return on the value added by negative samples as the count of positive samples rises.

Data quality matters in negative-aware training We sourced negative data from various models to investigate the impact of negative data quality in NAT. Specifically, we consider the data from GPT-3.5 as high-quality examples. In contrast, we generated 10k negative examples using a fine-tuned LLaMA-2-7B model to represent low-quality data. For our NAT process, we paired 2k positive examples with 10k negative examples. The outcomes, presented in Table 3, underscore the critical role of data quality in NAT. In the 2k positive sample setting, the improvement is -3.16 for low quality compared to $+8.74$ for high quality negative examples. Similarly in the 5k positive sample setting, the improvements are -6.20 and $+3.25$, respectively.

Negative samples play a similar role as positive samples To explore what the model learns from negative trajectories, we randomly sample 100 successful trajectories from the training set (as a dev

Positive	Negative	Average
Correct	Incorrect	63.55
Good	Bad	63.91
A	B	63.15
Random string 1	Random string 2	**64.04**
Incorrect	Correct	63.33

Table 4: Results for models trained on prompts with and without interpretability.

set) and measure the perplexity of models trained with 2k positive examples (not overlapping with the dev set) and varying numbers of negative examples. Figure 5 shows the change in perplexity as the number of negative data increases. The perplexity decreases as more negative data is included, which indicates the model learns to fit successful trajectories with knowledge from failed trajectories. However, this curve seems to be asymptoting towards the end, and there is still a large gap between the curve with 5k positives, which shows that some properties or knowledge from successful trajectories can never be learned from failed trajectories.

NAT works by differentiating positive and negative samples Finally, we explore the role of prompts during negative-aware training. More specifically, does the content of the prompt enable LLMs to learn differently from successful and failed trajectories, or simply differentiate these trajectories? We propose two sets of prompts. One set is prompts with interpretability, such as have the model generate a correct or incorrect trajectory. Another set is prompts without interpretability. For example, simply add different letters to the front of positive and negative trajectories. Table 4 shows the results of models trained with interpretable and uninterpretable prompts. Different prompts do not show a large difference in performance, indicating that the performance boost of NAT comes from simply differentiating positive and negative data.

5 Question Answering

5.1 Datasets

For question-answering tasks, we collected trajectories based on HotpotQA (Yang et al., 2018) and StrategyQA (Geva et al., 2021). HotpotQA is a Wikipedia-based question-answering dataset where each question requires several steps of reasoning with supporting passages. We use 4k examples from the training set to generate trajectories. StrategyQA is also a multi-step question-answering dataset but the reasoning steps are implicit. The answer to its question is either yes or no. It consists of 2,780 examples, of which 1k are the training set. We provide examples from both datasets in Appendix A.

Strategy	7B		13B	
	EM	F1	EM	F1
w/o Neg	27.44	36.41	27.04	36.95
NUT	28.04	40.96	28.24	42.47
NAT	28.80	41.37	28.44	42.45
NAT-2	**29.76**	**42.51**	**29.60**	**43.29**

Table 5: Results of LLaMA-2-7B and 13B on HotpotQA. All results are reported as the mean score of 5 runs. For the 7B model, we report results using 1,500 negative samples; for the 13B models, we use 2k negative samples.

Strategy	7B	13B
w/o Neg	55.40	53.40
NUT	62.40	61.80
NAT	**65.80**	**64.60**

Table 6: Results of LLaMA-2-7B and 13B on StrategyQA with 1000 positive samples and 500 negative samples.

Similar to math tasks, we generate three QA trajectories. As discussed in Section 4.3, the quality of negative samples is important for the effectiveness of NAT. For HotpotQA, we filter out trajectories that do not give an answer within a certain number of turns or with a zero f1 score. Finally, we obtain 2k unique positive samples, and 2k negative samples. However, we find that 2k examples is enough for performance to saturate, and that adding more negative samples causes a performance drop. Therefore, we set the number of HotpotQA positive examples to 500 in the following experiments. We evaluate the performance of 500 randomly-sampled examples from HotpotQA and StrategyQA.

5.2 Results

Tables 5 and 6 show the results on HotpotQA and StrategyQA. On HotpotQA, NAT improves performance by more than 2 points in EM and 6 points in f1 score, compared to no negative samples. Compared with NUT, NAT is still about 1 better on EM and f1. On StrategyQA, NAT achieves more than 8 and about 3 improvement compared to no negative samples and NUT, respectively.

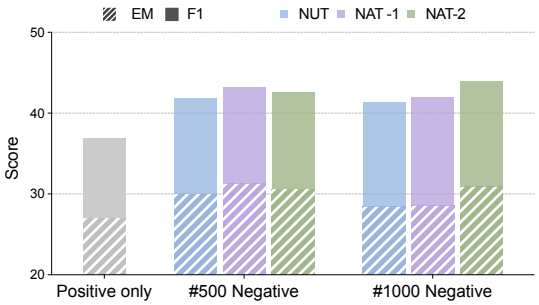

Figure 6: Performance of LLaMA-2-7B on HotpotQA with 500 positive examples and varying numbers of negative examples.

Strategy	GSM8K	ASDiv	SVAMP	MArith	Avg
w/o Neg	29.04	55.26	45.60	80.87	52.69
NUT	33.50	**61.69**	52.20	**86.41**	58.45
NAT	**36.24**	61.10	**53.90**	86.24	**59.37**

Table 7: LLaMA-2-7B model CoT results fine-tuned using 2k positive samples and 1.6k negative samples.

5.3 Analysis

Fine-grained NAT In addition to the EM score, each trajectory in HotpotQA has an f1 score, measuring the overlap between the predicted and gold answers. We take this as a fine-grained measurement of data quality, where a trajectory with a higher f1 score has better quality. In this way, we can differentiate trajectories based on quality by assigning different prompts. For example, the trajectory is labeled as *almost wrong* if its f1 score is smaller than 0.1, and another trajectory is *mostly correct* with an f1 score of 0.9. We denote this NAT with different prompting strategies as NAT-k, where k represents how many classes we divide the negative data into. For NAT-2, we take trajectories with f1 scores equal to 1.0 as positive, and assign different prompts for trajectories with f1 scores less than 0.4 and with f1 scores greater than 0.4 less than 1.0. It can be seen from Table 5 that the NAT-2 consistently outperforms NAT in all settings, indicating that negative samples associated with finer-grained info are more informative.

Fine-grained NAT learns more from negative samples We investigate how the performance changes with differing numbers of negative samples in Figure 6. When adding negative samples, the performance increases by a margin, consistent with Section 5.2. However, NAT achieves the best performance with only 500 negative samples, and its performance decreases when adding more negative samples. NAT-2, on the other hand, achieves the best performance with 1k negative samples, consistent with our hypothesis that fine-grained NAT is more beneficial to model training.

6 Chain-of-Thought Prompting

So far we have conducted all experiments on agent scenarios with `ReAct` prompting strategy. In this section, we conduct preliminary experiments to explore whether NAT works well with the Chain-of-Thought (CoT) prompting strategy (Wei et al., 2022). The key difference is that the agent receives an observation from the environment and then generates thought–action pairs, while CoT generates reasoning steps without observations.

We use GPT-3.5-0125 to generate CoT reasoning steps with three in-context examples on the GSM8k dataset. We then train the model with NAT. Table 7 shows the results with CoT prompting. NAT achieves a 6.68 improvement compared to no negative data training. NAT is still about 1 point higher compared to directly including negative samples (NUT). The results demonstrate that NAT is also applicable and effective for CoT training.

7 Conclusion

In this paper, we first demonstrated that LLMs can learn from failures when fine-tuned as an agent. On the basis of this finding, we propose NAT, a simple and effective method for integrating failed trajectories in fine-tuning agents. We conduct experiments on math and question-answering tasks, and show the superior performance of our method when training directly with positive or negative trajectories across tasks and model sizes. Our analysis finds that the quality of negative data is the key to the success of our method, and that models learn similar knowledge as what they learn from increased positive data (which is much more expensive to attain). We also demonstrated that our method is applicable and effective in Chain-of-Thought scenarios.

Negative-aware training is designed to be both agent-agnostic and reasoning strategy-agnostic, implying its compatibility with various agent strategies, including self-refinement and reflection. At the end of this paper, we demonstrated the effectiveness of NAT on Chain-of-Thought (COT) reasoning in mathematical tasks. Moving forward, we aim to assess the applicability and effectiveness of NAT across a broader spectrum of agent frameworks, strategies, and tasks.

8 Limitations

Although we have conducted extensive experiments to illustrate the effectiveness of our method, there are still limitations. First, similar to previous work in tool learning, our approach requires the ground truth label of the data, while in real scenarios, there is usually no ground truth label, which limits its application. Second, we do not experiment with fine-tuning our method on more diverse and powerful models (e.g. GPT-3.5) due to the time and budget limits.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *ArXiv*, abs/2310.05915.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Significant Gravitas. 2024. Autogpt. `https://github.com/Significant-Gravitas/AutoGPT`.

Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. 1961. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li. 2023. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data. *AAAI 2024*.

Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. 2024. From llm to conversational agent: A memory enhanced architecture with fine-tuning of large language models. *ArXiv*, abs/2401.02777.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Yuxian Gu, Hangliang Ding, Kai Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Shengqi Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents. *ArXiv*, abs/2308.03688.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch via self-planning. *ArXiv*, abs/2401.05268.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2019. Zero: Memory optimizations toward training trillion parameter models. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752. The Association for Computational Linguistics.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2023. Identifying the risks of lm agents with an lm-emulated sandbox. *ArXiv*, abs/2309.15817.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. https://api.semanticscholar.org/CorpusID:258833055.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Joseph Weizenbaum. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Michael Wooldridge. 1999. Intelligent agents. *Multiagent systems: A modern approach to distributed artificial intelligence*, 1:27–73.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Raghavi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. *ArXiv*, abs/2311.05657.

Yoheinakajima. 2024. Babyagi. https://github.com/Significant-Gravitas/AutoGPT.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *ArXiv*, abs/2310.12823.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Y. Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *ArXiv*, abs/2308.10144.

A Example

Figure 7 shows an example from HotpotQA, and Figure 8 shows an example from StrategyQA.

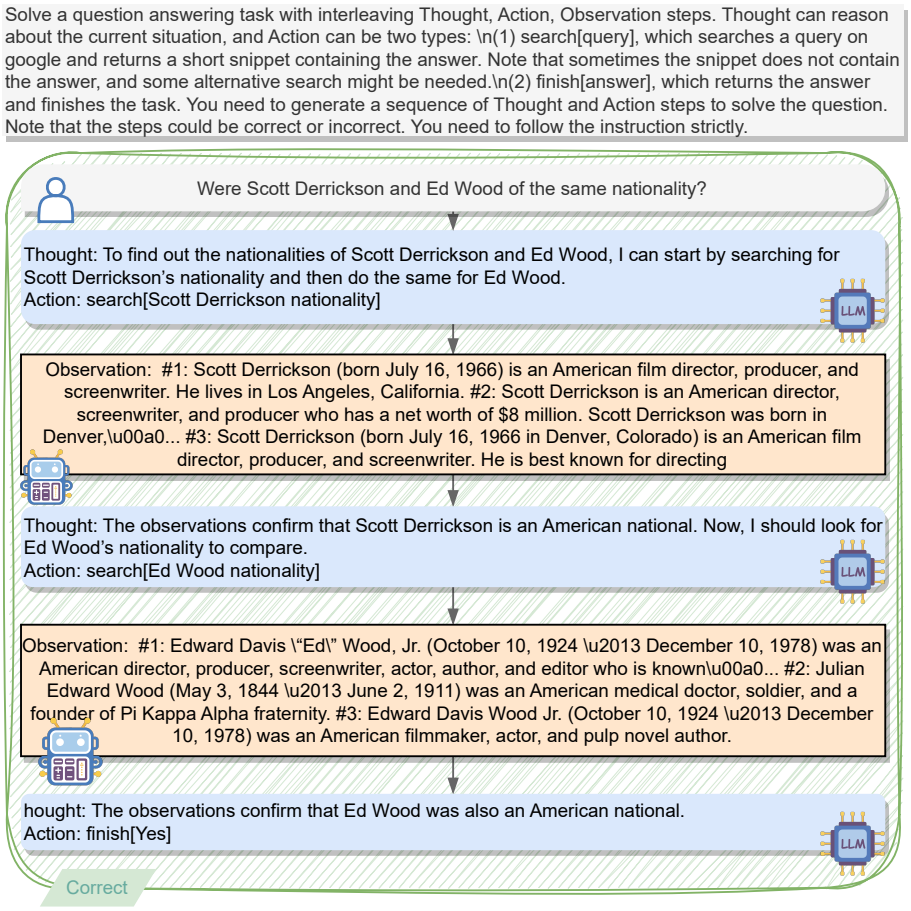

Figure 7: An example of a successful trajectory from HotpotQA

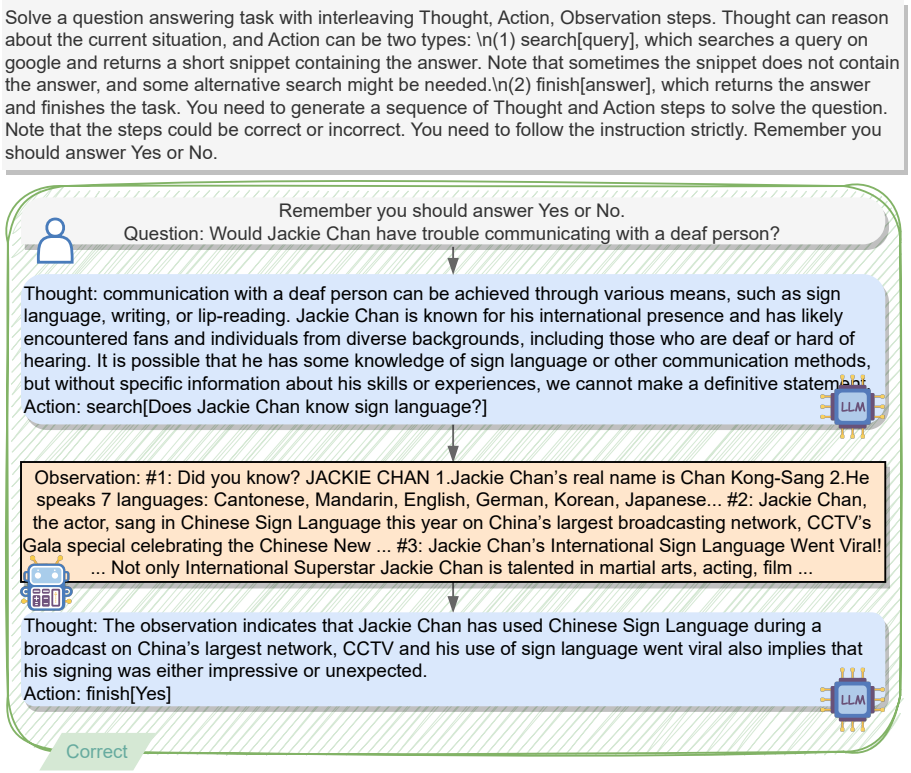

Figure 8: An example of a successful trajectory from StrategyQA